REVISTA DO PROGRAMA DE PÓS-GRADUAÇÃO EM EDUCAÇÃO MATEMÁTICA DA UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL (UFMS)

ISSN 2359-2842 Volume 14, número 36 – 2021 DOI: 10.46312/pem.v14i36.12121

Programação Intuitiva: em Busca de Compreensões

Intuitive Programming: in Search of Understandings

Renata Oliveira Balbino[1]

Marco Aurélio Kalinke[2]

Evandro Alberto Zatti[3]

Silvana Gogolla de Mattos[4]

Taniele Loss[5]

Marcelo Souza Motta[6]

[1] Doutoranda em Educação Matemática pelo Programa de Pós-Graduação em Educação em Ciências e em Matemática (PPGECM) da UFPR. Professora de Matemática da rede Estadual de ensino no Estado do Paraná. Membro do Grupo de Pesquisa sobre Tecnologias na Educação Matemática (GPTEM). E-mail: rebalbino@yahoo.com.br. ORCID: https://orcid.org/0000-0003-3402-3422.

[2] Doutor em Educação Matemática pela PUC-SP e pós-doutorado pela Universidade de Milão. É professor da UTFPR e membro do corpo docente do PPGECM da UFPR e do Programa de Pós-graduação em Formação Científica, Educacional e Tecnológica (PPGFCET) da UTFPR. Membro do GPTEM e do Grupo de Pesquisa em Inovações e Tecnologias na Educação (GPITENDUC). E-mail: marcokalinke23@gmail.com. ORCID: https://orcid.org/0000-0002-5484-1724.

[3] Doutorando em Educação Matemática pelo Programa de Pós-Graduação em Formação Científica Educacional e Tecnológica (PPGFCET) da UTFPR. Professor do ensino superior em cursos relacionados à Tecnologia da Informação. Membro do GPTEM. E-mail: evandro.zatti@live.com. ORCID: https://orcid.org/0000-0003-3123-1197.

[4] Doutoranda pelo Programa de Pós-Graduação em Educação em Ciências e em Matemática (PPGECM) da UFPR. Professora de Matemática da rede estadual de ensino do Estado do Paraná, atuando na Diretoria de Tecnologia e Inovação Educacional. Membro do GPTEM. E-mail: syl.mattos@gmaill.com. ORCID: https://orcid.org/0000-0002-6685-8638.

[5] Doutoranda e Mestre em Educação Matemática pelo Programa de Pós-Graduação em Formação Científica Educacional e Tecnológica (PPGFCET) pela UTFPR. Membro dos Grupos de pesquisa GPTEM e do GPITENDUC. Professora de matemática em Curitiba e acadêmica bolsista da UTFPPR. E-mail: tani_loss@hotmail.com. ORCID: https://orcid.org/0000-0002-0384-3260.

[6] Doutor em Ensino de Ciências e Matemática (UNCSUL). Professor do Departamento Acadêmico de Matemática (DAMAT) da UTFPR. Professor permanente do Programa do Programa de Pós-Graduação em Formação Científica, Educacional e Tecnológica (PPGFCET) da UTFPR e membro do corpo docente do PPGECM da UFPR. Coordenador a Especialização em Inovação e Tecnologias na Educação (INTEDUC); Líder do Grupo de Pesquisa em Inovações e Tecnologias na Educação (GPITENDUC) e membro do GPTEM. E-mail: msmotta@gmail.com. ORCID: https://orcid.org/0000-0001-5534-2735.

CC BY NC ND

RESUMO

Este artigo tem como objetivo a busca por ampliar compreensões acerca da “programação intuitiva”, para propor a sua definição no contexto educacional. Nesse intuito, realizou-se um estudo teórico e reflexivo ancorado em uma revisão literária de autores que tratam da temática investigada. Essa busca partiu do conhecimento dos fundamentos da programação de computadores para, na sequência, tratar da programação no âmbito educacional. Percebeu-se que a definição do termo “programação intuitiva” não é consenso entre os profissionais da informática e educadores. São apresentados alguns *softwares*, os quais possibilitam o uso de recursos de programação nos processos de ensino e de aprendizagem. Com base nas ferramentas observadas e nas formas de sua utilização, pode-se, então, propor uma definição para “programação intuitiva” como sendo uma linguagem de programação que leve em consideração as características de similaridade, visualização e acessibilidade.

PALAVRAS-CHAVE: Programação de Computadores. Processos Educativos. Educação Tecnológica. Intuição. Software.

ABSTRACT

This article aims to seek to broaden understandings about “intuitive programming”, to propose its definition in educational context. To this end, a theoretical and reflective study was carried out based on a literary review by authors dealing with the investigated theme. This search started from the knowledge of the fundamentals of computer programming to, next, deal with programming in the educational scope. It was noticed that the definition of the term “intuitive programming” is not a consensus among computer professionals and educators. Some software, which enable the use of programming resources in the teaching and learning processes. Based on the tools observed and the ways in which they are used, it is then possible to propose a definition for “intuitive programming” as a programming language that takes into account the characteristics of similarity, visualization and accessibility.

KEYWORDS: Computer Programming. Educational Processes. Technology Education. Intuition. Software.

Introdução

A presença das Tecnologias Digitais (TD) em nossa sociedade vem transformando os meios de comunicação e informação em diversos setores. Uma das suas características é a possibilidade de auxiliar na realização de tarefas simultâneas, com maior rapidez e sem a necessidade da presença física. A partir da década de 1970, uma nova fase na automação industrial com linhas de produção flexíveis e máquinas industriais com controles digitais foi consequência da utilização de computadores. Desde então, o uso de dispositivos eletrônicos, computadores e redes de comunicação de dados vem se consolidando na sociedade (LÉVY, 2010).

A partir das mudanças sociais resultantes da presença de TD, as proposições de Lévy (1993) podem contribuir na busca de compreensões acerca da construção do conhecimento ancorada na sua utilização no âmbito educacional. Surge, entre outras, a necessidade de desenvolver habilidades cognitivas que contribuam com os processos educacionais por meio da utilização de ferramentas computacionais. Um exemplo dessa contribuição pode ser a criação de projetos com o uso de aplicativos

e, para tanto, pode-se explorar processos de ensino e de aprendizagem que façam uso de recursos de programação.

Nesta direção, Raabe, Zorzo e Blikstein (2020) consideram a atividade de programação como recurso metodológico educacional que pode promover momentos de compreensão e construção do pensamento computacional. Em concordância, Resnick (2006) afirma que ao desenvolver o pensamento computacional os estudantes colocam em prática habilidades de raciocínio lógico e abstração. Tais autores pontuam que quando a programação é utilizada para fins educacionais, os comandos podem ser mais simples que os utilizados por um programador profissional. Logo, entende-se que não é necessário um aprofundamento no entendimento da sintaxe quando destinado a área da educação.

Uma vez abstraídos os processos da programação profissional, no contexto educacional a programação pode ser inserida como uma atividade que seja a mais intuitiva possível. É neste panorama que surge o termo “programação intuitiva”, que tem sido utilizado por autores como Oliveira (2019), Kalinke e Motta (2019) e Santos, Lourenço, Júnior e Barbosa (2008).

Numa busca pela literatura, constatou-se que tal termo não possui uma definição posta e amplamente aceita. Logo, percebe-se uma carência de aprofundamento sobre o assunto. Neste viés, entende-se que particularidades quanto as compreensões sobre a temática possam colaborar com discussões sobre conceitualização de “programação intuitiva”. Nesta perspectiva, objetiva-se a busca por ampliar compreensões acerca da “programação intuitiva”, para propor a sua definição no contexto educacional.

Para resolver o problema proposto, realizou-se uma pesquisa qualitativa ancorada em uma revisão literária em estudos de autores como Papert (1980), Santos *et al*. (2008), Resnick *et al*. (2009) e Zatti (2017), que tratam da temática a ser investigada. Realizou-se um estudo de cunho teórico que foi “dedicado a reconstruir teoria, conceitos, ideias, ideologias, polêmicas, tendo em vista, em termos imediatos, aprimorar fundamentos teóricos” (DEMO, 2000, p. 20). Logo, as sínteses interpretativas são resultado de análises e interpretações dos conceitos teóricos identificados na literatura.

Na intenção de buscar compreensões sobre “programação intuitiva”, parte-se do conhecimento dos fundamentos e conceitos gerais da programação de computadores para, na sequência, tratar da programação no âmbito educacional. São apresentados os *softwares Scratch*, *App Inventor* e o *Hot Potatoes* que podem

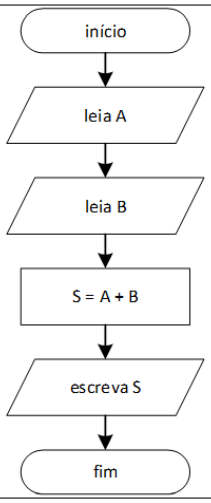
ser utilizados na educação e são indicadas proposições das compreensões acerca da temática, culminando numa definição para o termo “programação intuitiva”.

Programação de Computadores

A sequência lógica de passos para resolver um determinado problema ou executar uma tarefa, é o que se chama, em programação, de algoritmo. Segundo Zatti (2017), um algoritmo pode ser notado de três formas, a saber: narrativa, gráfica ou pseudocódigo. A forma narrativa acontece em linguagem natural (humana), que é passível de interpretação e não oferece um grau aceitável de formalismo. Já a notação gráfica é feita com o uso de fluxograma, no qual diferentes figuras denotam funcionalidades específicas. O pseudocódigo é a maneira que mais se aproxima de uma linguagem de programação, pois traz a sequência de comandos (escreva, leia, compute, entre outros) e manipulação de variáveis escritas em uma linguagem próxima à natural, porém com formalismo e sintaxe definidos. A Figura 1 apresenta um mesmo algoritmo sob as três diferentes formas de notação.

Figura 01 - Formas de notação de um algoritmo

Narrativa (linguagem natural)	**Gráfica** (fluxograma)	**Pseudocódigo** (português estruturado)

Sejam A e B dois números, informados pelo usuário, calcule a soma entre eles e apresente o resultado		`Algoritmo "somadoisnumeros"`
		`Var`
		`  a, b, s: inteiro`
		`Inicio`
		`  leia (a)`
		`  leia (b)`
		`  s <- a + b`
		`  escreva (s)`
		`Fimalgoritmo`

Fonte: elaborado para a pesquisa

Quando um algoritmo é convertido para uma sequência que possa ser executada por um computador (ou equipamento digital), ele passa a ser um programa. Nesta perspectiva, a criação de uma sequência de comandos para um computador, utilizando uma linguagem específica para este fim, é denominada programação.

Os computadores digitais realizam o processamento por meio de sequências binárias (zeros e uns), nas quais os dados são representados e os comandos são baseados em decisões proposicionais, que se resumem a um resultado verdadeiro

ou falso. Por este motivo, os primeiros computadores eram programados diretamente através de chaves e botões “ligados ou desligados”. Mesmo com a evolução tecnológica, o princípio não mudou e os computadores ainda trabalham com sequências binárias, que são conhecidas como linguagem de máquina. Sendo assim, para programar um computador, é necessário converter as sequências de comandos que idealizamos em um algoritmo para a linguagem que o computador entende. É neste ponto que entram as linguagens de programação.

Para programar um computador, o algoritmo deve ser expresso em uma linguagem de programação disponível para o sistema que se deseja programar (como, por exemplo, C++, Java ou C#), obedecendo à sintaxe e semântica próprias. A sequência de comandos é digitada em um arquivo de texto, chamado código fonte, que deve passar por um processo de compilação. A compilação do código fonte irá convertê-lo para a linguagem de máquina, que resultará nas sequências binárias executadas pelo computador.

Sebesta (2016) sugere que as linguagens de programação podem ser divididas em três paradigmas (categorias): imperativo, funcional e lógico. No paradigma imperativo a programação acontece por meio de comandos que devem ser obedecidos, tais como “faça isso, faça aquilo”. As linguagens funcionais, por sua vez, resolvem os problemas mediante aplicação de funções para parâmetros fornecidos. Por fim, a programação lógica pressupõe o uso de regras e as suas combinações para apresentar um resultado ou inferência. A Figura 2 ilustra estes três diferentes paradigmas.

Figura 02 - Os diferentes paradigmas de programação

Paradigma Imperativo

Código do programa em Linguagem C++:

```
void somadoisnumeros()
{
    int a, b, s;
    std::cin >> a;
    std::cin >> b;
    s = a + b;
    std::cout << s;
}
```

Execução/uso do programa:

```
2
3
5
```

Paradigma Funcional

Código do programa em LISP

```
;; Dobro do número

(defun dobro (N)
  "Calcula 2 vezes n"
  (* 2 N))
```

Execução/uso do programa:

```
USER(1): (dobro(5))
10
```

Paradigma Lógico

Código do programa em Prolog

```
conhece(joao,tecnologia).
conhece(maria,tecnologia).
conhece(joao,matematica).
conhece(maria,matematica).
```

Execução/uso do programa:

```
?- conhece(joao,tecnologia)
yes.
?- conhece(joao,X)
X = tecnologia
X = matematica
?- conhece(Y,matematica)
Y = joao
Y = maria
```

Fonte: elaborado para a pesquisa

A maioria das linguagens de programação comerciais, utilizadas para criação de programas de manipulação de dados ou construção de aplicativos de uso geral, está sob o paradigma imperativo. Neste tipo de programação, o objetivo é alcançado por meio de comandos de entrada e saída, cálculo e manipulação de dados.

Considerando a divergência entre a linguagem de máquina e a linguagem humana, também é possível classificar as linguagens de programação de acordo com seu grau de abstração (ZATTI, 2017). Linguagens que fazem uso de símbolos mnemônicos e instruções muito próximas da linguagem de máquina são classificadas como de baixo nível. Em contrapartida, aquelas que possuem sintaxe mais próxima da linguagem natural são classificadas como de alto nível.

Para um programador profissional são exigidos conhecimentos específicos da área de programação e o computador é a sua ferramenta de trabalho. Sendo assim, é essencial partir do domínio das noções básicas de *hardware* e *software*, ter raciocínio lógico e saber construir algoritmos eficazes para que se possa criar programas utilizando diferentes linguagens, cada uma com as suas regras, em diferentes níveis de abstração e sob diversos paradigmas. A escolha da linguagem depende dos objetivos traçados para cada projeto.

Nesta perspectiva, a compreensão que se busca para a programação intuitiva, na educação, estaria em uma categoria superior à das linguagens de programação de alto nível, pois se aproximaria ainda mais, devido às suas simbologias, à linguagem natural. De forma análoga, no passado o uso de mnemônicos foi substituído por uma simbologia inteligível (TANENBAUM; AUSTIN, 2013), o que dá mostras da constante evolução da área.

A programação no âmbito educacional

Segundo Resnick *et al.* (2009), a metodologia pautada na programação pode estimular o desenvolvimento cognitivo das crianças que atuam na busca de soluções mentais para problemas e desafios propostos. A programação, contudo, pode ser entendida como uma das componentes das TD, cuja inserção nos processos educacionais encontra sustentação e fundamentação teórica em Tikhomirov (1981), Lévy (1993, 2010) e Kenski (2011).

No caso específico da Educação Matemática, as TD se fazem presentes e estão sendo exploradas há algumas décadas. Um olhar atento a ela, e dedicado à busca por compreensões sobre a inserção e o uso de linguagens de programação, pode mostrar que elas se fazem presentes desde o início desta trajetória, e a acompanham até aqui. Borba, Silva e Gadanidis (2018) propõem uma organização cronológica da utilização das TD no contexto da Educação Matemática, que pode ser ampliada para todos os contextos educacionais, dividida em quatro fases.

A primeira delas é caracterizada pelo uso da linguagem Logo[7]. A principal perspectiva teórica sobre o uso pedagógico do Logo é a teoria do Construcionismo de Seymour Papert (1980), que enfatiza a construção do conhecimento baseada em ações concretas desenvolvidas por meio do uso do computador, que sejam de interesse do usuário. As atividades, desenvolvidas no Logo, possibilitam que o estudante possa investigar, levantar hipóteses, testá-las e refiná-las, no objetivo da construção de seu próprio conhecimento (PAPERT, 1980).

Por meio dessa linguagem o usuário pode formar sequências de comandos específicos que possibilitem a execução de uma atividade. Esse ambiente propõe uma ruptura com um modelo educacional tradicional no qual professor e aluno são vistos, respectivamente, como detentor e receptor do conhecimento. O Logo

[7] Linguagem de programação criada no final da década de 60, idealizada por Seymour Papert. A palavra “Logo” é utilizada para se referir tanto à linguagem em si quanto ao *software* no qual a linguagem é empregada.

apresentou as primeiras características do que, futuramente, evoluiria em busca de se tornar uma linguagem de programação que pode remeter à noção de intuitiva, uma vez que para seu uso não eram necessários conhecimentos avançados de programação. Ela exigia conhecimentos básicos de comandos que, uma vez fornecidos ao *software*, faziam com que ele desse como retorno ao usuário movimentos de uma tartaruga que transformava os comandos em ações. Como exemplo é possível criar uma imagem que represente a figura geométrica de um quadrado no ambiente logo com o comando “repita 4 [pf 50 pd 90]”, conforme ilustra a Figura 3.

Figura 03 - Quadrado desenhado por meio de programação Logo

Fonte: elaborado para a pesquisa

Para construir o quadrado, a tartaruga que realiza os movimentos na tela irá se deslocar para frente 50 pixels[8] (pf 50), traçando um dos lados do quadrado. Em seguida, irá rotacionar seu corpo 90 graus para a direita (pd 90), posicionando-se para traçar o lado subsequente. Esta sequência, repetida 4 vezes, completa a figura geométrica do quadrado. Já se percebia, no Logo, a intenção de uso de recursos de programação no ensino sem, contudo, a necessidade de dominar uma linguagem de programação tal como se exige dos programadores profissionais. O comando necessário é simples, ainda que se possa discutir se é intuitivo.

A linguagem Logo propõe colocar a criança no comando de um robô, ou de uma representação dele por meio da tela do computador. A representação simbólica de uma tartaruga como modelo de robô pode ser compreendida como uma forma de utilização da arte na comunicação das ideias. Nesta proposta, estudantes e professores podem ser considerados artistas que elaboram suas próprias

[8] Pixel é a menor unidade para composição de uma imagem digital.

estratégias para criar ou resolver projetos de seu interesse. O Logo proporciona ao sujeito a imersão em um micromundo de interatividades com o *software*.

A segunda fase das tecnologias é marcada pela produção de diversos *softwares* educacionais que possibilitaram a exploração de diferentes práticas didáticas e pedagógicas. No âmbito da Matemática se destaca a utilização do *Winplot*[9], *Cabri Géomètre*[10], *Maple*[11], entre outros. Para Borba, Silva e Gadanidis (2018, p. 27) "esses *softwares* são caracterizados não apenas por suas interfaces amigáveis, que exigem pouca ou nenhuma familiaridade com linguagens de programação, mas principalmente pela natureza dinâmica, visual e experimental".

Percebe-se novamente que os recursos utilizados para as práticas pedagógicas buscam contribuir com os processos educacionais minimizando a necessidade do aprendizado de linguagens de programação específicas. Os *softwares* mencionados são de natureza dinâmica, visual e experimental, com interfaces que não exigem conhecimento específico de linguagem de programação para serem exploradas. Usuários que dominassem as linguagens de programação de origem dos *softwares* poderiam explorá-los de forma diferenciada, mas isto não era necessário para o uso dos recursos básicos, explorados pelos usuários em geral.

A presença da internet é a principal característica da terceira fase das tecnologias. Com ela, novas formas de comunicação e informação foram disponibilizadas e, nesta fase, além do termo "TI", que se refere às Tecnologias de Informação, consolidou-se o termo "TIC", que se refere às Tecnologias da Informação e Comunicação. De acordo com Borba, Silva e Gadanidis (2018), a difusão de ambientes virtuais de aprendizagem possibilitou interações coletivas, de formas síncronas e assíncronas, entre os usuários.

Surgem, neste contexto, *softwares* que permitem a criação e disponibilização de conteúdos na internet sem a necessidade de domínio do código HTML (*Hyper Text Markup Language*, em português: Linguagem de Marcação de Hipertexto), por exemplo. No princípio da internet, quando o hipertexto se tornou um padrão de navegação e acesso, para disponibilizar alguma informação era necessário escrevê-la em HTML, que era a linguagem padrão reconhecida pelos navegadores. Com a

[9] *WinPlot* é um programa para gerar gráficos de 2D e 3D a partir de funções ou equações matemáticas.
[10] O *Cabri Géomètre* é um *software* de geometria dinâmica.
[11] *Maple* é um sistema algébrico computacional comercial de uso genérico.

popularização da rede, surgiram diversas soluções que transformavam formatos como “doc” em HTML, além de programas específicos para a criação de sites sem que o domínio da linguagem HTML fosse necessário. Nos dias atuais, disponibilizar informações na rede, seja por meio de sites pessoais, redes sociais ou por blogs, pode ser considerado algo corriqueiro e dominado por muitos usuários, incluindo alunos da Educação Básica.

A quarta fase de uso das tecnologias teve início a partir das melhorias das conexões, incluindo a velocidade de transmissão de dados, e do aprimoramento da quantidade e dos tipos de recursos com acesso à internet, que possibilitou a transformação da comunicação *on-line*. Além da difusão do termo “TD”, relativo às Tecnologias Digitais, esta fase é caracterizada pelo uso de recursos tais como *softwares* educacionais, redes sociais, vídeos e objetos de aprendizagem nos processos educacionais.

Borba, Silva e Gadanidis (2018, p. 41) consideram essa fase “um cenário exploratório, fértil ao desenvolvimento de investigações e à realização de pesquisas”. Para os autores, esta é a fase vivida atualmente, na qual o uso das TD busca compreender as características de multimodalidade, telepresença, interatividade, internet em sala de aula, uso de artes na comunicação de ideias matemáticas, produção e compartilhamento *on-line* de artefatos digitais. Ela agrega aos processos educacionais, recursos tecnológicos que exploram as linguagens de programação por meio de interfaces amigáveis e nas quais o domínio de uma linguagem específica não seja necessário. É assim com a criação de objetos de aprendizagem, por exemplo, que tem sido explorada em muitas atividades educacionais (KALINKE; MOTTA, 2019; ELIAS, 2018; MEIRELES, 2017; BALBINO, 2016).

O avanço dos recursos existentes levou à incorporação de *softwares* que fazem a tradução dos códigos de programação sem que o usuário precise dominá-los. Novamente, usuários que dominem os recursos específicos eventualmente podem criar e explorar possibilidades desconhecidas aos usuários comuns. Surgem assim, novamente, exemplos da diferença entre o programador profissional e os usuários de linguagens de programação com finalidades educacionais.

Possibilidades de *softwares* para programação na educação

É possível aprender diferentes linguagens de programação com o auxílio de projetos que tenham como objetivo estimular e incentivar o estudo da programação.

Essa forma não profissional de programar permite a resolução e proposição de problemas com a criação de estratégias e o estímulo do pensamento criativo.

Segundo Tikhomirov (1981) a reorganização do pensamento, como consequência do uso do computador, não apenas expande a capacidade da atividade existente, mas permite despontar um novo estágio de pensamento criativo. Caracterizado essencialmente pela originalidade de uma ideia, o pensamento criativo diferencia-se pelas formas de resolução de um dado problema pelo indivíduo com o uso do computador.

Dentre os diversos *softwares* que se enquadram no contexto educacional de uso de linguagens de programação é possível indicar, como já citado, o *Scratch*, o *App Inventor* e o *Hot Potatoes*. O *Scratch* e o *App Inventor* atualmente estão sob a responsabilidade de pesquisadores do Instituto de Tecnologia de Massachusetts (MIT), mesmo local em que Papert desenvolveu a linguagem Logo. O *Hot Potatoes* foi criado por pesquisadores do Centro de Computação em Humanidades e Mídia da Universidade de Vitória, no Canadá.

No bojo deste trabalho optamos por estes *softwares*, por eles serem indicados por diversos autores (ELIAS, 2018; ROCHA, 2018; MEIRELES, 2017; ZOPPO, 2017; CAPELIN, 2015) como sendo viáveis para o uso do que se tem chamado de "programação intuitiva", foco de nossa busca por compreensões.

No *Scratch* e no *App Inventor*, as ideias básicas das programações exploradas em ambos guardam similaridades. O que os difere, fundamentalmente, é que enquanto o primeiro de destina essencialmente à criação de projetos em computadores, o segundo é voltado à criação de *apps*[12] para dispositivos móveis, como *tablets* e *smartphones*. Ambos, contudo, apresentam a mesma ideia, de programação desenvolvida com o uso de blocos lógicos constituídos de códigos prontos. Eles podem ser considerados evoluções da linguagem Logo no sentido de terem sido desenvolvidos depois dela e contribuírem com o uso de programação na educação, mas não como uma evolução da linguagem em si.

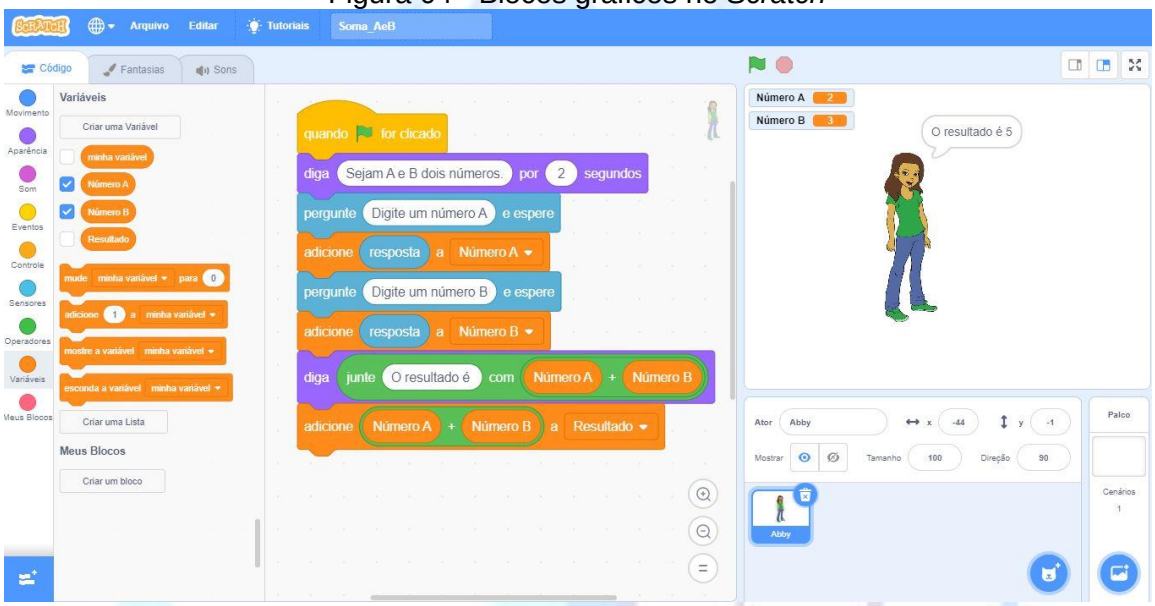
O *Scratch* é um *software* gratuito dotado de uma linguagem de programação visual que oferece uma interface gráfica baseada em blocos de comandos. Foi desenvolvido a partir de investigações e do aperfeiçoamento das linguagens e ambientes de programação para jovens pela equipe coordenada por Mitchel Resnick

[12] O termo *app*, abreviatura de *application* (do inglês: aplicação ou aplicativo) é utilizado popularmente para se referir especificamente aos aplicativos para dispositivos móveis, tais como, *smartphones* e os *tablets.*

do MIT. Apesar deste *software* ser recomendado especialmente para crianças e jovens com faixa etária compreendida entre 8 e 16 anos, ele também é utilizado por pessoas de outras faixas etárias.

A escrita de um programa nessa linguagem é realizada por meio do encaixe de blocos lógicos. Os comandos são separados por cores, de acordo com a sua função, e a sequência de instruções pode ser modificada a qualquer momento do desenvolvimento. O *Scratch* permite a construção de objetos digitais que podem combinar gráficos, animações, textos, músicas e outros elementos multimídia, conforme ilustrado na Figura 4.

Figura 04 - Blocos gráficos no *Scratch*



Fonte: elaborado para a pesquisa

Os ícones são disponibilizados na parte esquerda da tela, a programação é feita no centro e, à direita, tem-se o resultado do projeto. Para realizar a programação, os blocos devem ser arrastados da área esquerda para a área central e encaixados conforme o algoritmo desejado. No exemplo ilustrado pela figura 4, a programação configurada para o ator “Abby” determina a soma entre dois números A e B digitados pelo usuário.

Eloy, Lopes e Angelo (2017, p. 1) afirmam que “o *Scratch* torna a prática da programação acessível a diferentes públicos, permitindo que jogos e histórias interativas digitais sejam criados e compartilhados por meio da conexão de blocos intuitivos e próximos da língua falada”. A similaridade do idioma e a acessibilidade dos ícones de forma visual possibilitam a criação de atividades por meio da interatividade do usuário com a plataforma.

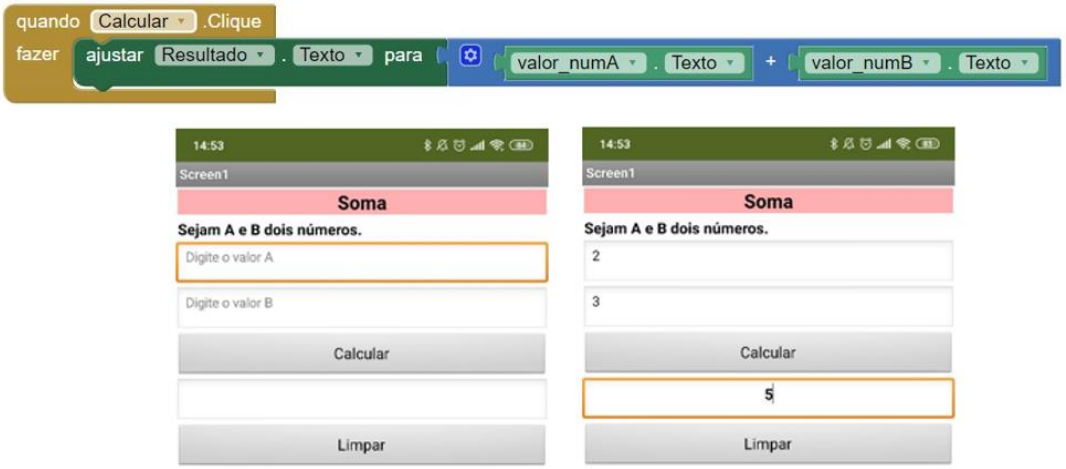
A sua proposta, como recurso educacional, busca minimizar o distanciamento entre a evolução tecnológica atual e a fluência tecnológica dos cidadãos. Nesse contexto, os usuários podem deixar de ser consumidores de tecnologia para se tornarem criadores. No ambiente escolar, a utilização do *Scratch* pode contribuir com o desenvolvimento da fluência tecnológica, uma vez que os computadores, considerados meios de expressão criativa, podem colaborar com os processos educacionais. Para Resnick (2006) a fluência computacional deve ser trabalhada no mesmo nível da leitura e da escrita.

Com o *App Inventor* é possível criar, de forma livre e gratuita, *apps* para dispositivos móveis. Ele também possibilita o desenvolvimento de soluções que sejam adaptadas de forma a explorar a interatividade com o usuário, sem a necessidade do domínio de uma linguagem de programação específica. O *App Inventor*:

> Abriu caminho na educação computacional pesquisando sobre maneiras didáticas de se construir aplicativos para o sistema operacional móvel *Android*. O sistema, baseado na *web*, permite que qualquer usuário com experiência básica em computação crie uma aplicação capaz de rodar no sistema operacional por meio de programação visual em blocos. (TORRES; AROCA; BURLAMAQUI, 2014, p. 273).

A estrutura de programação do *App Inventor* também usa o conceito de programação em blocos, que são encaixados segundo códigos preexistentes e que não permitem o encaixe de blocos que não executem um comando viável. Tal estrutura pode ser observada na Figura 5, que apresenta o encaixe de blocos que determina o resultado da soma entre dois números digitados pelo usuário, e as telas de visualização desta programação no dispositivo móvel.

Figura 05 - Blocos gráficos no *App Inventor*



Fonte: elaborado para a pesquisa

Segundo Elias (2018):

> A partir do trabalho por meio de blocos de encaixe, padrão do *App Inventor*, [...], professores da Educação Básica ou Ensino Superior, que tiverem pouco ou nenhum conhecimento em programação, podem desenvolver seus projetos de forma intuitiva. As peças utilizadas na programação do *software* assemelham-se a peças de um quebra-cabeça e são de fácil manipulação. Basta que o usuário selecione a opção que deseja trabalhar, arrastá-la para o visualizador e ir encaixando os itens escolhidos. (ELIAS, 2018, p. 69).

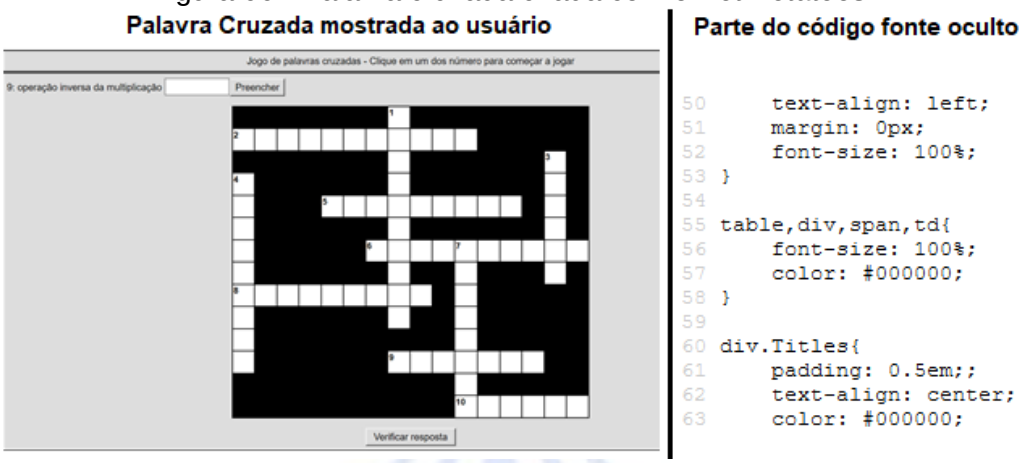
Pode-se perceber que o *Scratch* e o *App Inventor* possuem similaridades, uma vez que ambos propõem a atividade de programação por meio do encaixe de blocos gráficos com códigos prontos ou que podem ser editados conforme os objetivos dos usuários. Além disso, os blocos de comandos são acessíveis em sua tela inicial e apresentados no idioma similar ao do usuário. Tais recursos possibilitam a visualização e a materialização de um programa desenvolvido sem a necessidade de um conhecimento de linguagem de programação específica para tal.

É importante que não se confunda programação por blocos com “programação intuitiva”. Certamente o modelo de programação por encaixe de blocos com programações predefinidas pode ser um dos modelos de programação intuitiva. Entretanto, há outros modelos que não usam estes blocos e que também podem ser explorados intuitivamente. É o caso do *Hot Potatoes*, por exemplo, que usa o código de marcação HTML ao invés dos blocos.

O *Hot Potatoes* é um *software* educativo gratuito, interativo e dinâmico que possibilita o desenvolvimento de cinco tipos diferentes de atividades: *JCross* para elaboração de palavras cruzadas; *JMix* para exercícios de ordenação de palavras em uma frase; *JCloze* para exercícios de texto com espaços em branco para preencher; *JQuiz* para questionários de múltipla escolha, seleção múltipla, verdadeiro/falso, ou de resposta curta; e *JMatch* para exercícios de associação de pares.

Embora os nomes das diferentes atividades sejam apresentados em inglês, o *software* permite ao usuário escolher outro idioma. O *Hot Potatoes* possibilita a criação de atividades por meio da interatividade do usuário com a plataforma. Um exemplo de atividade elaborado no *JCross* está apresentado na Figura 6:

Figura 06 - Palavra cruzada criada com o *Hot Potatoes*



Fonte: elaborado para a pesquisa

O código fonte desta atividade possui um total de 1555 linhas de comandos. Para o usuário não profissional, entretanto, elas são ocultadas e não precisam ficar visíveis. Ele pode construir a atividade de forma visual, em uma interface que faz a construção do código de modo automático.

Programação Intuitiva: caminhos para a compreensão

Tendo como referência as ferramentas apresentadas anteriormente e suas formas de programações, pode-se buscar algumas compressões sobre a programação intuitiva. Santos *et al.* (2008) definem “programação intuitiva” como “uma programação na qual não é necessário um aprofundamento no entendimento da linguagem utilizada” (SANTOS *et al*., 2008, p. 1). Etimologicamente, a palavra intuição vem do latim *intuitus*, que significa considerar, ver interiormente ou contemplar (HOUAISS, 2020). De acordo com Dewey (1925), a intuição possui algumas qualidades:

> Esses "sentimentos" têm uma eficiência de operação que é impossível para o pensamento corresponder. Até nossas operações mais altamente intelectualizadas dependem deles como uma "margem" pela qual guiar nossos movimentos inferenciais. Eles nos dão nosso senso de retidão e injustiça, sobre o que selecionar, enfatizar e acompanhar, e o que deixar cair, zombar e ignorar entre a multidão de significados incipientes que se apresentam ... Essas qualidades são o material das "intuições". (DEWEY, 1925, p. 244).

Nesta concepção, pode-se considerar o conhecimento intuitivo como parte inerente ao ser humano, como uma capacidade inconsciente de propor e resolver problemas ou para a tomada de decisões. As conclusões baseadas no raciocínio intuitivo podem ser resultado de ações involuntárias uma vez que não foram utilizados padrões ou planejamentos para sua resposta.

> Intuição é a experiência subjetiva de um processo principalmente inconsciente, rápido, lógico e inacessível à consciência que, dependente da exposição ao domínio ou espaço problemático, é capaz de extrair com precisão contingências probabilísticas. (LIEBERMAN, 2000, p. 111).

Com o objetivo de compreender o termo “programação intuitiva”, considera-se a programação como uma prática educacional que envolve a resolução de desafios, raciocínio múltiplo e desenvolvimento do pensamento crítico. Uma atividade exploratória pode contribuir com a resolução e busca de soluções de forma intuitiva. A atividade de programação pode colaborar com a prática de fazer suposições, pois é uma ação que possibilita a pesquisa e a contextualização, particularmente na resolução de problemas.

Para além das possibilidades da programação e ampliando as discussões sobre “programação intuitiva”, percebe-se, a partir dos exemplos apresentados, que uma ferramenta computacional baseada na intuição possui preocupações com três pilares: a similaridade, a visualização e a acessibilidade.

No que se refere à similaridade, indica-se que é um ambiente que apresente ferramentas com estrutura de linguagem semelhante à do usuário. Dessa forma, o foco principal da ação de programar se detém na criação e no desenvolvimento do produto e não na aprendizagem de uma nova linguagem. Essa indicação vai ao encontro do trabalho proposto pela equipe de Hopper (1952), quando criou o compilador, baseado na sua língua nativa, a língua inglesa. Até os dias atuais, o uso de compiladores é imprescindível na transposição da linguagem humana para linguagem do computador, mesmo que eles estejam invisíveis aos programadores não profissionais.

Nos três ambientes explorados neste trabalho (*Scratch, App Inventor* e *Hot Potatoes*), se observa que a linguagem utilizada nas interfaces é a linguagem do usuário, na sua língua nativa, para uma grande quantidade de idiomas.

Com relação ao *App Inventor*, baseando-se na funcionalidade deste ambiente, Pokress e Veiga (2013) apresentam que ele fornece uma metáfora intuitiva de programação. Por exemplo: ao criar um *app* que envia e recebe mensagens de texto, existe disponível na plataforma, na seção interface do usuário, o botão “CaixaDeTexto” que pode ser usado para executar tal função. A relação entre o que se deseja programar e o que é disponibilizado na plataforma, também é identificada no ambiente de edição dos blocos, visto que ele fornece dicas gráficas ao usuário. Se o objetivo for determinar a soma entre dois números, existem blocos que indicam esta operação, conforme ilustrado na Figura 7.

Figura 07 - Blocos que indicam a ação de somar dois números

Fonte: elaborado para a pesquisa

Esta figura exemplifica também a possibilidade de o usuário se concentrar na lógica de programação, ao invés de se preocupar com a sintaxe da linguagem de programação utilizada. Para Pokress e Veiga (2013) o ambiente de programação do *App Inventor*, pode diminuir o nível de frustração que é muitas vezes identificado quando se realiza programação baseada somente em linguagem de texto.

Sobre o segundo pilar, a visualização, ressalta-se a importância do programador (que pode ser profissional ou não), poder identificar seus resultados a qualquer momento do desenvolvimento do projeto. Este pilar possibilita a seleção e a correção de eventuais erros de sintaxe do programa. Do ponto de vista cognitivo, pensando na linguagem Logo, ensinar a tartaruga exige da criança uma reflexão sobre a tarefa; visualizar os passos que constituem esse processo é concretizar uma abstração (Papert, 1980). Uma plataforma com viés intuitivo possibilita que seus processos sejam visualmente identificáveis. No caso dos *softwares Scratch* e *App Inventor*, por exemplo, os comandos são separados por cores e os blocos se encaixam, mesmo quando a sequência não faça, a priori, sentido na programação. No caso do *Hot Potatoes* o ambiente já é apresentado ao usuário na sua interface final, que não exige a utilização dos códigos HTML pelos usuários.

Tomando como base a pesquisa de Mulholland (1997), a visualização de um programa consiste no uso das mais variadas técnicas a fim de representar o próprio programa em execução. A preocupação visual pode colaborar com a interatividade uma vez que oferece uma forma de comunicação entre o equipamento e o usuário, e tem o cuidado de transmitir mensagens de forma clara, que evitem ambiguidades. Um ambiente interativo de aprendizagem oferece ao usuário meios interativos de visualização.

Outra característica desejada numa plataforma com tendência intuitiva é a acessibilidade, que vai além da transposição de barreiras arquitetônicas. Ela se refere a que os comandos básicos da interface sejam acessíveis e estejam dispostos por meio de botões/ícones, oportunizando que o usuário escolha os comandos disponíveis na plataforma. A presença de ícones representativos, que apresentam uma estrutura de linguagem similar à do usuário, contribui com a acessibilidade do programa.

Uma plataforma acessível permite o desenvolvimento de projetos sem a necessidade de uma formação específica. Está relacionada ao fato de possibilitar condições para a utilização, total ou assistida, da linguagem de programação, isto é, o usuário não precisa saber todos os comandos necessários para programar naquele ambiente. Basta identificar os elementos disponíveis na plataforma, que apresentam dicas ao usuário. Assim, intuitivamente, a partir do seu entendimento (interpretação), o usuário seleciona o elemento (bloco gráfico, por exemplo) que deseja para criar seu projeto.

As opções presentes nos blocos de encaixe do *Scratch* e *App Inventor* e nas telas do *Hot Potatoes* podem ser exemplos para ilustrar o que se compreende por ambientes com acessibilidade.

Conclusão

Este artigo teve como objetivo a busca por ampliar compreensões acerca da “programação intuitiva”, para propor a sua definição no contexto educacional. Nessa intenção, realizou-se uma síntese do que se apresenta na literatura e as principais características de três plataformas de programação. Aprofundar os estudos acerca desse assunto trouxe reflexões sobre quais seriam as características que poderiam determinar a intuitividade na programação.

A busca por compreensões é uma atividade que abre possibilidades e diferentes perspectivas. Na busca de características presentes em ambientes que façam uso da “programação intuitiva”, percebe-se que eles apresentam algumas características comuns bem delimitadas, a saber:

- não necessitam o domínio de uma linguagem específica de programação, ainda que este domínio possa ser benéfico aos usuários que o possuam;

- trabalham com a construção de projetos e resolução de problemas sem a necessidade de digitação de códigos fontes, mas utilizando blocos de códigos preexistentes ou interfaces que os traduzam para o usuário;
- apresentam ferramentas com estrutura de linguagem semelhante à do usuário (similaridade);
- possibilitam que seus processos sejam identificáveis, visualizando os resultados em qualquer momento do desenvolvimento do projeto (visualização);
- possibilitam a utilização da linguagem de programação por meio de ícones/blocos que podem auxiliar na construção do projeto/programa (acessibilidade).

Os ambientes utilizados como exemplos ao longo do texto apresentam as características indicadas. Destaca-se, como já indicado anteriormente, que estes ambientes não são os únicos, mas ilustram satisfatoriamente o que entendemos por ambientes que utilizam a “programação intuitiva” em atividades educacionais. Com base no exposto, pode-se propor uma definição para programação intuitiva como sendo uma linguagem de programação destinada à construção de projetos educacionais em ambientes computacionais que não necessitem o domínio de uma linguagem de programação específica e que apresentem características de similaridade, visualização e acessibilidade.

Frente a observação e descrição das plataformas *Scratch*, *App Inventor* e *Hot Potatoes*, identificaram-se aspectos que contribuíram na proposta de definir o que se pode entender por “programação intuitiva”. Observa-se que a intuitividade de uma plataforma pode ser alcançada pela combinação de três características: similaridade, visualização e acessibilidade. Elas estão relacionadas à forma como o usuário interage com uma plataforma para a programação em interface gráfica.

Propõe-se uma definição de “programação intuitiva” que leve em consideração características de similaridade, visualização e acessibilidade, sem a necessidade do domínio de uma linguagem de programação. Ressalta-se que não é necessário dominar a sintaxe de uma linguagem específica, de modo que um ambiente seja explorado sem que códigos de programação sejam diretamente utilizados.

A contribuição desse trabalho também corrobora com a utilização de plataformas intuitivas no propósito de suprimir ou diminuir as dificuldades na aprendizagem de uma linguagem própria de programação. Isso permite que a

preocupação do usuário assente na construção dos algoritmos que permitam solucionar problemas.

Com base nestas considerações propõe-se uma possível compreensão para o que se tem apresentado como programação intuitiva. Espera-se que novas compreensões possam surgir e serem discutidas, em busca de avanços para a incorporação destes recursos nas atividades educacionais, uma vez que a definição proposta neste trabalho está aberta a novos estudos, tais como os que tratem de subcategorias de programação intuitiva, por blocos ou visual, por exemplo.

Referências

DEMO, Pedro. **Metodologia do conhecimento científico**. São Paulo: Atlas, 2000.

BALBINO, Renata Oliveira. **Os objetos de aprendizagem de Matemática do PNLD 2014**: uma análise segundo as visões construtivista e ergonômica. 139 p. Dissertação (Mestrado em Educação Matemática) – Universidade Federal do Paraná, Curitiba, 2016.

BORBA, Marcelode Carvalho; SILVA, Ricardo Scucuglia Rodigues da; GADANIDIS, George. **Fases das tecnologias digitais em Educação Matemática.** 2 Ed; 2 reimp. Belo Horizonte: Autêntica Editora, 2018.

BRACKMANN, Christian Puhlmann. **Desenvolvimento do pensamento computacional através de atividades desplugadas**. 226 p. Tese (Doutorado) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2017.

CAPELIN, Alcione. **O ensino de funções na lousa digital a partir do uso de um objeto de aprendizagem construído com vídeos.** 147 p. Dissertação (Mestrado em Educação Matemática) – Universidade Federal do Paraná, Curitiba, 2016. Disponível em: https://acervodigital.ufpr.br/handle/1884/41864. Acesso em 28 out. 2020.

DEWEY, John. **Experience and nature. La Salle**, IL: Open Court, 1925.

ELIAS, Ana Paulade Andrade Janz. **Possibilidades de utilização de** *smartphones* **em sala de aula:** construindo aplicativos investigativos para o trabalho com equações do 2º grau. 135 p. Dissertação (Mestrado em Formação Científica, Educacional e Tecnológica) - Universidade Tecnológica Federal do Paraná, Curitiba, 2018. Disponível em: https://repositorio.utfpr.edu.br/jspui/handle/1/3897. Acesso em 28 out. 2020.

ELOY, Adelmo Antônio da Silva; LOPES, Roseli de Deus; ANGELO, Isabela Martins. Uso do *Scratch* no Brasil com objetivos educacionais: uma revisão sistemática. **Revista Novas Tecnologias na Educação**, v.15, n. 1, p. 1, jul. 2017. Disponível em: https://seer.ufrgs.br/renote/article/view/75164. Acesso em: 28 out. 2020.

HOPPER, Grace Murray. The Education of a Computer*. In***: Proceedings of the 1952 ACM National Meeting**, Pittsburgh: 243–249. Disponível em: https://dl.acm.org/doi/pdf/10.1145/609784.609818. Acesso em: 28 out. 2020.

HOUAISS, A. **Dicionário Online de Português.** Disponível em: https://houaiss.uol.com.br/pub/apps/www/v5-2/html/index.php#0. Acesso em: 28 out. 2020.

KALINKE, Marco Aurélio; MOTTA, Marcelo Souza. **Objetos de Aprendizagem:** pesquisas e possibilidades na Educação Matemática. Campo Grande: Editora Life, 2019.

KENSKI, Vani Moreira. Em direção a uma ação docente mediada pelas tecnologias digitais. In: BARRETO, R. G. (Org.) **Tecnologias educacionais e educação à distância**: avaliando políticas e práticas. Rio de janeiro: Quartet, 2001.

KENSKI, Vani Moreira. **Educação e tecnologias**: o novo ritmo da informação. 8° ed. São Paulo: Papirus, 2011.

LÉVY, Pierre. **As tecnologias da inteligência:** o futuro do pensamento na era da informática. São Paulo: Editora 34, 1993.

LÉVY, Pierre. **Cibercultura**. 3° Ed. Rio de Janeiro: Editora 34, 2010.

LIEBERMAN, Matthew Dylan. Intuition: A Social Cognitive Neuroscience Approach. *In* **Psychological Bulletin**. vol. 126, n. 1, 109-137, 2000. Disponível em: https://doi.org/10.1037/0033-2909.126.1.109. Acesso em 28 out. 2020.

MEIRELES, Tatiana Fernandes. **Desenvolvimento de um objeto de aprendizagem de matemática usando o** *Scratch***: da elaboração à construção**. 168 p. Dissertação (Mestrado em Educação Matemática) – Universidade Federal do Paraná, Curitiba, 2017. Disponível em: https://acervodigital.ufpr.br/handle/1884/56109. Acesso em 28 out. 2020.

MULHOLLAND, Paul. Incorporating Software Visualization into Prolog teaching: a challenge, a restriction, and an opportunity. *In*: **Proceedings of ICLP'97 Postconference workshop on logic programming environment**, 14, 1997, Leuven. Proceedings of ICLP'97 Postconference Workshop on Logic Programming Environment. Mit Press, p. 33 – 42, 1997.

OLIVEIRA, Elaine Cristina de. **Contribuição da programação intuitiva nos anos iniciais do ensino fundamental.** Monografia de conclusão de curso. Especialização em inovação e tecnologias na educação. Universidade Tecnológica Federal do Paraná, 2019. Disponível em: http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/15329/1/CT_INTEDUC_I_2019_19.pdf . Acesso em 28 out. 2020.

PAPERT, Seymour. **Mindstorms**: children, computers and powerfull ideas. New York, Basic Books. 1980.

POKRESS, Shaileen Crawford; DOMÍNGUEZ VEIGA, José Juan. MIT *App Inventor*: enabling personal mobile computing. **Proceedings of Programming for Mobile and Touch**, PRoMoTo, 2013. Disponível em: https://arxiv.org/pdf/1310.2830.pdf. Acesso em: 28 out. 2020.

RAABE, Aandré; ZORZO, Avelino; BLIKSTEIN, Paulo. (org). **Computação na educação básica**: fundamentos e experiências. Porto Alegre: Penso, 316 p.2020.

RESNICK, Mitchel *et al* Scratch: Programming for All. **Communications of the ACM**, 52(11), 60-67. Disponível em: https://doi.org/10.1145/1592761.1592779. Acesso em: 28 out. 2020.

RESNICK, Mitchel. **Computer as Paintbrush**: Technology, Play, and the Creative Society. 2006. Disponível em: http://llk.media.mit.edu/papers/playlearn-handout.pdf. Acesso em: 28 out. 2020.

ROCHA, Flavia Sucheck Mateus. **Análise de projetos do** *Scratch* **desenvolvidos em um curso de formação de professores.** 135 p. Dissertação (Mestrado em Educação Matemática) – Universidade Federal do Paraná, Curitiba, 2018. Disponível em: https://acervodigital.ufpr.br/handle/1884/59437. Acesso em: 28 out. 2020.

SANTOS, Celso Eduardo dos; LOURENÇO, Jose Luis; JÚNIOR, Alderico Rodrigues de Paula; BARBOSA, Luis Filipe Wiltgen. Desenvolvimento de um sistema baseado em blocos para programação intuitiva em microcontroladores. *In*: **XII Encontro Latino Americano de Iniciação Científica e VIII Encontro Latino Americano de Pós-Graduação**, 2008. P.1. Paraíba. Anais eletrônicos. Disponível em: http://www.inicepg.univap.br/cd/INIC_2008/anais/arquivosINIC/INIC0372_01_A.pdf. Acesso em: 28 out. 2020.

SEBESTA, Robert Wayne. **Concepts of Programming Languages**. 11th global ed. Harlow: Pearson, 2016.

TANENBAUM, Andrew Stuart. **Structured Computer Organization**. London: Pearson Education Inc, 2013.

TIKHOMIROV, Oleg Konstantinovich. The psychological Consequences of Computerization. In Wertsch, J. V. (Ed.). **The Concept of Activity in Soviet Psychology**. New York: M. E. Sharpe Inc. p. 256- 278, 1981.

TORRES, Victor Paiva; AROCA, Rafael Vidal; BURLAMAQUI, Aquiles Filgueira. Ambiente de programação baseado na web para robótica educacional de baixo custo. *In*: **Holos**, vol. 5. 2014, p. 252-259 Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte Natal, Rio Grande do Norte, 2014. Disponível em: https://doi.org/10.15628/holos.2014.1902. Acesso em 28 out. 2020.

ZATTI, Evandro Alberto. **Programação orientada a objetos**. Curitiba: Fael, 2017.

ZOPPO, Beatriz Maria. **A contribuição do** *Scratch* **como possibilidade de material didático digital de Matemática no ensino fundamental I**. 135 p. Dissertação (Mestrado em Educação Matemática) – Universidade Federal do Paraná, Curitiba, 2017. Disponível em: https://acervodigital.ufpr.br/handle/1884/53394. Acesso em: 28 out. 2020.

Submetido em novembro de 2020.

Aceito em outubro de 2021.